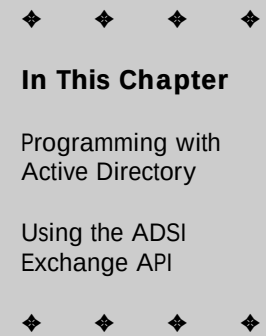
CHAPTER 20

Active Directory Services Advanced Topics



In This Chapter

Programming with Active Directory

Using the ADSI Exchange API

Active Directory is a very complex product and requires that you have knowledge of both COM and database programming if you want to work with it. It's going to take most programmers a long time to get up to speed with this new technology and even longer before administrators will feel comfortable with it. In short, you'll want to spend time researching Active Directory and its capabilities before you start writing your first application.

The previous chapter looked at the architectural and usage details of Active Directory. In this chapter, we look at the programming interfaces for Active Directory in the form of the Active Directory Service Interfaces (ADSI). Make sure you understand the underlying theory behind Active Directory before you continue with this chapter.

The first section of this chapter looks at Active Directory itself. We'll answer questions such as, "What are the programming considerations for this environment?" It's important to understand that Active Directory is a multidiscipline product. A small change in the database schema, the addition of an MMC snap-in, or the addition of an application component can greatly affect the usage environment, so you need to exercise care when making changes. Obviously, the new nature of this product won't help you when it comes time to make choices about which portions of Active Directory to implement in your next application.

The final section looks at the COM components that Microsoft provides for working with Active Directory in the form of ADSI version 2.5. This section of the chapter assumes that you're using the latest version of ADSI available at the time of writing.

I provide information on where to download this version. Besides the standard ADSI COM interfaces, we also look at what you need when working with Exchange. Active Directory is designed to provide a fairly complete interface that allows it to work with Exchange. You can use this capability to perform tasks such as maintaining a centralized list of company contacts.

Working with the Active Directory

Active Directory is a very complex part of Windows 2000. Not only are there the database elements, but there's also interaction with other directory services, management issues, and even a new set of programming features implemented by the COM components that make up ADSI. We've talked about most of these issues in the previous chapter and will continue the discussion of ADSI later in this chapter. What we haven't talked about are the programming considerations for Active Directory itself. That we're managing a large distributed database changes the entire picture for the programmer. There are things that you need to consider that other kinds of applications don't even touch. For example, how do you create a connection to an object that may reside in another city or country? Obviously, because Active Directory replicates all of the data for the entire database on each domain controller does make access faster, but that replication can work against you when it comes to recording changes in the schema or object attributes.

You aren't alone when it comes to Active Directory. Even before Microsoft released Windows 2000, people started creating Web sites devoted to working with Active Directory. One site that looks very promising at the time of this writing is Active Directory Networks at `http://activedirectory.net/`. This Web site promises detailed information about Active Directory. If you need a Web site that contains more information at an administrator level in order to manage the components you create, then check out the Exploring Active Directory site at `http://www.servin.com/explorin8.htm`.

There are three important programming considerations that we'll talk about in this chapter. In the first section, we look at security. Of course, security is always a concern, especially when you're talking about the configuration data for a large organization. Considering the kinds of information that Active Directory makes available, security isn't just a priority; it's a major part of the programming picture.

Gaining access to objects is another concern, so that's the Active Directory programming consideration we look at in the second section. Microsoft calls this process *binding*. In sum, you're creating a connection to the object for the purpose of manipulating the data that it contains. Binding a directory services object is a requirement because of the various kinds of directory services that Active Directory supports. The object within the directory appears as a standard COM object to the programming language. In addition, the kind of COM object that the programming language sees doesn't vary by directory service, which makes it easy to use the same code no matter which directory service you want to access.

A task that you'll likely perform when working with directory objects is modifying the attributes of groups and users. The third section will tell you how to work with these two important object types. Even if your application doesn't directly modify users or groups, you need to interact with them to see what rights they possess and to potentially change those rights so that the user or group can use your application.

Security

We've discussed some of the security basics for Active Directory in Chapter 17. Make sure you read that chapter before proceeding in this one. Like the rest of Windows 2000, Active Directory relies on a system of access tokens and security descriptors to ensure the security of each object on the machine. If you haven't read that discussion yet, you may want to take the opportunity to do so now because knowledge of the basic security features that Windows 2000 provides is important for this section of the chapter.

Gaining access to Windows 2000 doesn't necessarily mean that you have automatic access to all of Active Directory. During the process of binding, the requesting object provides an access token that describes the object's rights to various parts of the system. If the requesting object has sufficient rights, then a connection between the requesting object and the Active Directory object is made. From that point on, the requesting object has access, but nothing else. Each time the requesting object asks Active Directory to perform a task, the requesting object's access token is compared to the Active Directory object security descriptor. If the requesting object has sufficient rights, Active Directory performs the required task. This process goes on continually to ensure that the requesting object always acts within the confines of its rights.

So, what happens with child objects? The parent controls the rights inherited by a child object. Depending on how security is set up, a requesting object may have more or less access to child objects than to the parent. In fact, if the requesting object doesn't really require access to the child object, the parent may not grant any rights to use it at all. (The parent still retains control over the child object.) Contrast this behavior to the standard behavior where rights automatically flow from top to bottom (if you have access to a top-level object, the access must be specific revoked at lower levels), and you'll see that Active Directory actually provides more stringent security than the rest of Windows 2000.

There's also a matter of delegation. Active Directory assumes that only a select few administrators have complete access to the objects in the directory. However, delegating control allows workgroup managers and others to work with select portions of the directory database. You can add delegation as part of the setup for your application or ask the administrator to add this later. Delegation can occur at any level of the directory tree, except at the directory level. Let's go through a quick delegation scenario so that you can see how it works. You'll probably want to create a temporary Active Directory object that you can delete once you see the process in action. The example uses an organizational unit called Temp for demonstration purposes.

Try This

1. Right-click an object and choose the Delegate Control option from the context menu. You'll see the Delegation of Control Wizard dialog box.
2. Click Next. You'll see an Active Directory Folder dialog box like the one shown in Figure 20-1. Active Directory is asking what you want to delegate control of. It will always start with the domain or organizational unit. However, you can specify an object within the folder as well. When writing an application, you'd need to specify the globally unique identifier (GUID) of the object that will get delegated.

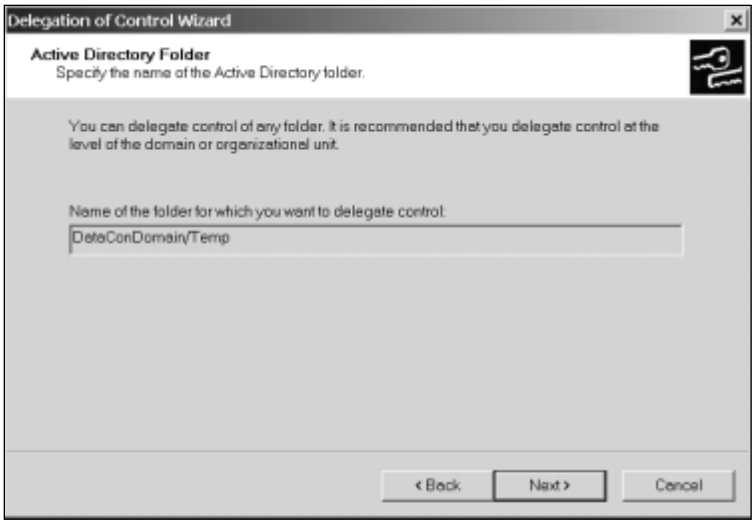

Delegation of Control Wizard
Active Directory Folder
Specify the name of the Active Directory folder.
You can delegate control of any folder. It is recommended that you delegate control at the level of the domain or organizational unit.
Name of the folder for which you want to delegate control:
DataConDomain/Temp
< Back
Next >
Cancel

Figure 20-1: Because we started the example by right-clicking on an organizational unit, Active Directory assumes we want to delegate control at that level.

3. Click Next. You'll see a Group or User Selection dialog box like the one shown in Figure 20-2. As with any other security dialog boxes, you click the Add button to add users or groups to the list. The sample dialog box has had Everyone group added to it for explanation purposes — normally, this dialog box starts out blank. If you were adding users for delegation purposes within an application, you'd need to create an ACE, then place the GUID of the Active Directory object that you wanted to delegate within the ACE using the put_ObjectType( ) method of the IADsAccessControlEntry interface. We'll talk more about the interfaces associated with security under Active Directory later in this section.
4. Add one or more groups and users, and then click Next. You'll see a Predefined Delegations dialog box like the one shown in Figure 20-3. This is where you determine what rights get delegated to the user or group that you

chose in the previous dialog box. You can choose between predefined task-oriented rights, or you can create new tasks of your own. When programming, you'll set these options using an ADS_RIGHTS_ENUM data structure. You'll place this structure within the ACE using the AccessMask field.

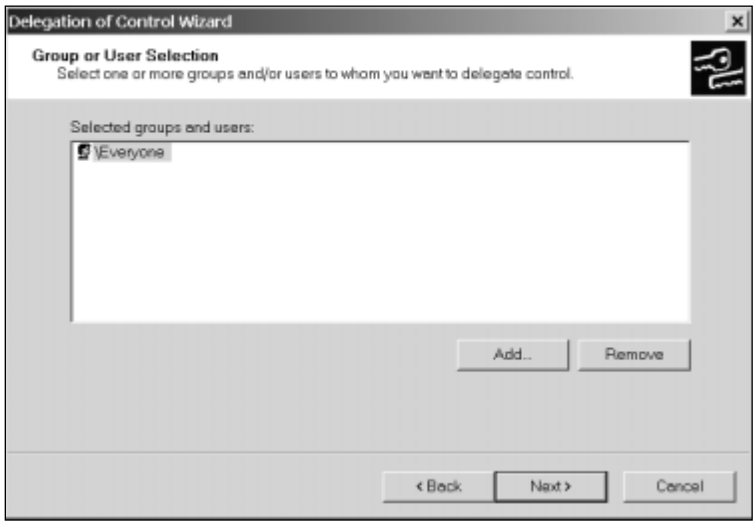


Figure 20-2: You need to add groups and users to this dialog box to delegate control of the object.

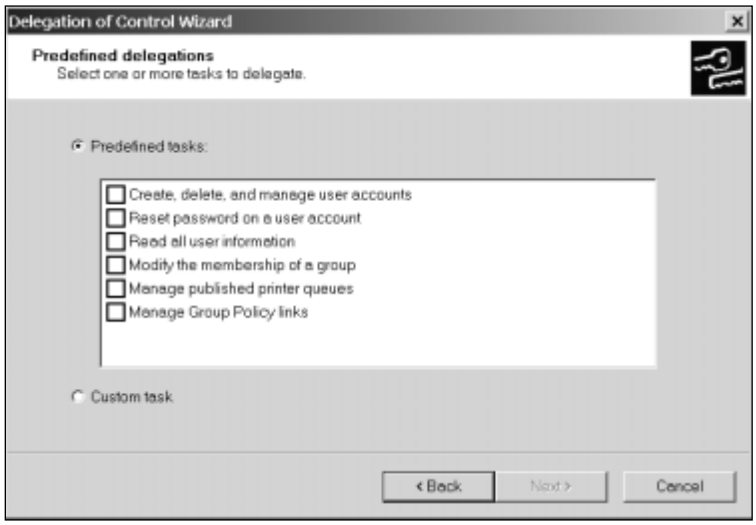

Delegation of Control Wizard
Predefined delegations
Select one or more tasks to delegate.
Predefined tasks:
Create, delete, and manage user accounts
Reset password on a user account
Read all user information
Modify the membership of a group
Manage published printer queues
Manage Group Policy links
Custom task
< Back
Next >
Cancel

Figure 20-3: Active Directory uses a task-oriented approach to granting rights for accessing a directory object.

5. Choose one or more predefined tasks, then click Next. You'll see a Completing the Delegation of Control Wizard dialog box like the one shown in Figure 20-4. This dialog box provides a good summary of the three things that you need to do to delegate authority over Active Directory objects: choose an object, add groups and users, and then define the rights that you want to delegate. If you were writing an application, you'd use the AddAce( ) method of the IADsAccessControlList interface to add the ACE to the list of ACEs for the object.

Delegation of Control Wizard
Completing the Delegation of Control Wizard
You have chosen to delegate control of objects in the following Active Directory folder:
DateConDomain/Temp
The groups, users, or computers to which you have given control are:
\Everyone
You selected the following predefined tasks to delegate:
Read all user information
< Back
Finish
Cancel

Figure 20-4: Delegating authority to work with a directory object is a three-step process.

MMC initially hides many Active Directory details from you. To see the change we've just made, create a temporary user in the organizational unit. Right-click the Temp organization unit and choose the View➪Advanced Features option. You'll see a lot of new objects that didn't appear before. Right-click the temporary user you created and choose Properties. Select the Security tab of the user Properties dialog box, then click the Advanced button. You'll see an Access Control Settings dialog box similar to the one shown in Figure 20-5. Notice that this dialog box shows the new permission that we added for the Everyone group. Interestingly, this is the only dialog box where you can see this information. Microsoft needs to work on this problem because it's possible right now for an administrator to assign rights that a user may not need and because of the difficulty of determining who has what rights, no one will see the security breach (at least not until someone breaks in or audits the accounts).

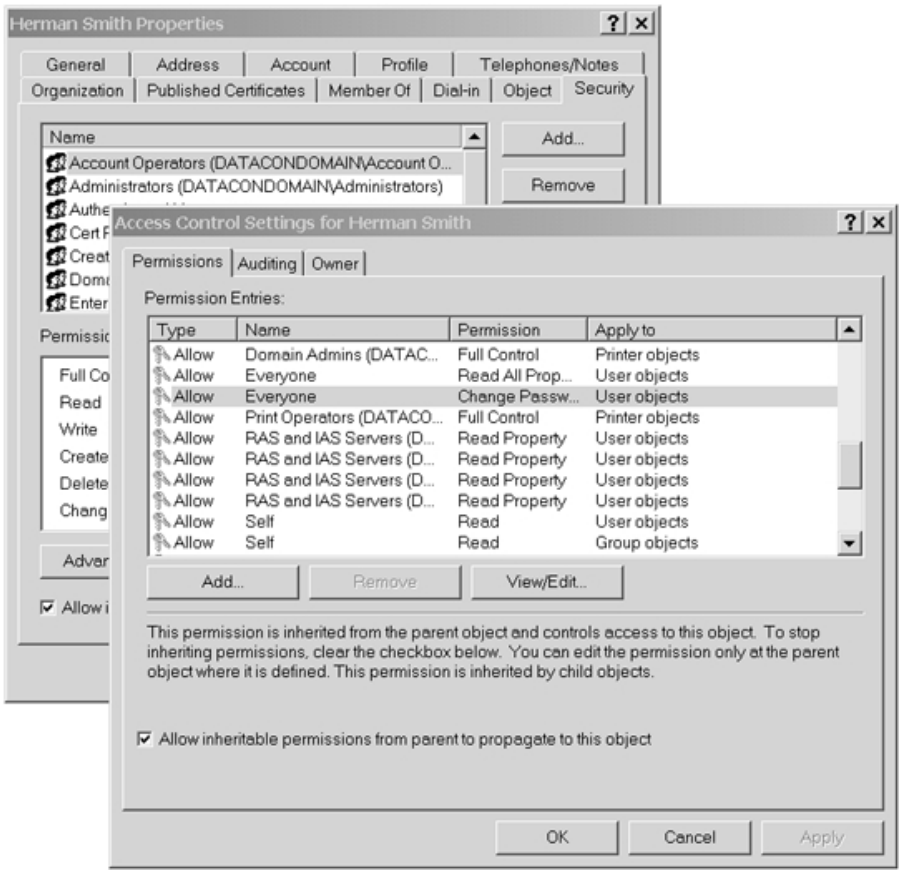

Herman Smith Properties
General
Address
Account
Profile
Telephones/Notes
Organization
Published Certificates
Member Of
Dial-in
Object
Security
Name
Account Operators (DATACONDOMAIN\Account O...
Administrators (DATACONDOMAIN\Administrators)
Add...
Remove
Access Control Settings for Herman Smith
Permissions
Auditing
Owner
Permission Entries:
Type
Name
Permission
Apply to
Allow Domain Admins (DATAC... Full Control Printer objects
Allow Everyone Read All Prop... User objects
Allow Everyone Change Passw... User objects
Allow Print Operators (DATACO... Full Control Printer objects
Allow RAS and IAS Servers (D... Read Property User objects
Allow RAS and IAS Servers (D... Read Property User objects
Allow RAS and IAS Servers (D... Read Property User objects
Allow RAS and IAS Servers (D... Read Property User objects
Allow Self Read User objects
Allow Self Read Group objects
Add...
Remove
View/Edit...
This permission is inherited from the parent object and controls access to this object. To stop inheriting permissions, clear the checkbox below. You can edit the permission only at the parent object where it is defined. This permission is inherited by child objects.
Allow inheritable permissions from parent to propagate to this object
OK
Cancel
Apply

Figure 20-5: The Access Control Settings dialog box allows you to determine what rights users and groups really have to an object.

Now that you understand what will go on during the delegation process, let's look at the security interfaces in a little more detail. Table 20-1 contains a list of the three security-related interfaces and describes their purpose.

All three of these interfaces include dispatch methods that allow you to get the number and IDs of type information descriptions, obtain a description of the object's programmable interface, and invoke methods within the object.

Table 20-1
Active Directory Security Interfaces

Interface	*Description*
IADsAccessControlEntry	Allows the calling application to gain access to individual ACEs within an object. The interface methods allow access to the access mask, ACE type, flags, object type, inherited object type, and trustee information.
IADsAccessControlList	Allows access to one of the ACLs that contain the ACEs used to access an object. The interface methods allow you to work with the ACL revision number, determine the number of ACEs that it contains, add or remove ACEs, copy the ACL, and enumerate the ACL contents.
IADsSecurityDescriptor	Provides access to the security descriptor for a directory service object. There are interface methods that allow you to access the revision number and owner and group information, and work with both the SACL and DACL.

Binding

At the very simplest level, binding is a process of finding an Active Directory object, requesting permission to access it, and then connecting to it. Once you have a reference to an object, you have access to any information, methods, or attributes that you're authorized to work with. So, there are two problems that any application you create has to solve. The first is finding the object in the first place. If you don't know where the object is, then you can't ask for access to it. The second is gaining access to the object once you've found it. ADSI provides two functions and one interface that you can use to perform this work:

- ADsGetObject( )
- ADsOpenObject( )
- IADsOpenDSObject::OpenDSObject( )

Of these three, the two methods are the preferred way to access a directory services object if you're using Visual C++ because they require less code and fewer processing cycles. The ADsGetObject( ) method is the one that you'll use if you want to access the object using the name and password of the person currently logged into the system. If you need to access the object using an alternative name and password, then you'll want to use the ADsOpenObject( ) method because it takes additional arguments that allow you to specify a name and password. You'll also use ADsOpenObject( ) if you want to encrypt the data moving between your application and the directory services object, or if you want to specifically bypass

authentication, so that you can access the object using the Everyone or Guest accounts. In all cases, you'll receive an indirect pointer to the requested object interface once the method succeeds.

You need to specify the name of the object that you want to access in some way. Active Directory currently allows you to use one of two methods as shown below.

```
GC://<Host Name>/<Object Name>
LDAP://<Host Name>/<Object Name>
```

Except for the access method, both of these examples are the same. You need to specify the name of the host for the object, which includes the fully qualified DNS name for the container, along with the name of the object that you want to access within that container. The host name is optional, in many cases, but including it's a good idea if you want to ensure fast access. The object name can include a distinguished name, canonical name, object GUID, or a SID. The use of GC allows you to access the object using entries in the global catalog, whereas LDAP allows you to access the object using that protocol. Of the two, GC normally provides the fastest access, but LDAP is the most flexible when it comes to multiple directory service scenarios.

ADSI supports serverless binding of objects, which means that it will find the object using the directory services tree, rather than looking for the object on a specific server. The use of serverless binding means that your code won't break if an object gets moved from one machine to another. In most cases, you want to avoid using a specific sever name to access an object. Use rootDSE as your server reference in place of a specific server name, which is a well-known and reliable location on every directory services tree. The rootDSE will always know the current location of the domain container, schema container, configuration container, and other server-specific information.

In the previous paragraph, I mentioned something called the global catalog. The global catalog is a special namespace that contains information from all of the domains in the forest. The global catalog is unique in a number of ways. For one thing, only a domain controller can act as a repository for the global catalog. The global catalog is also more of a search database rather than the definitive database of objects on the network. It contains one reference for every object, but not all of the attributes for those objects. Only the attributes that are marked for inclusion within the global catalog get included, which means that you get only a partial picture of the object if you look at it using the global catalog. As a result, the global catalog makes a good place to look for the object you need and to gain a reference to it, but you need to look through the directory services database to gain access to the full object. Nonautomation clients, such as Visual C++, can implement this two-step search-and-access method using two special interfaces: IDirectorySearch and IDirectoryObject. The first interface is used to search the global catalog to gain a reference to the desired object, whereas the second is used to directly access the object itself.

The hierarchical format of Active Directory allows you to perform two other kinds of object binding as well. You can bind to the parent of an object using the get_Parent( ) method of the IADs interface. Likewise, you can use the GetObject( ) method of the IADsContainer interface to retrieve a child object using a relative path. These two features allow you to move up and down the Active Directory as you would a doubly linked list. In addition, because you're maintaining a connection during the whole procedure, you don't have to pass your credentials to Active Directory again, which improves application performance. Active Directory does, however, check the existing access token against the security descriptor of any object you attempt to access.

The final binding topic we need to discuss is domain trees. You can access the entire forest using the global catalog as we talked about earlier in the section. However, this means searching every tree within the forest for the object you need. If you already know which tree contains the object that you want to access, searching the entire forest may be a waste of time. Fortunately, you can use the NetEnumerateTrustedDomainsEx( ) function to obtain a list of all of the trusted domain trees within the forest. Once you have this list, you can use it to find a particular tree and gain access to its root. At this point, you can search the tree for the object that you want to access.

Managing Users and Groups

Users and groups form the two most common kinds of objects that you'll find in any directory service. After all, the whole purpose of creating a network is to allow its users to share both resources and data. Using groups allows the network administrator to manage large groups of users with relative ease. Groups allow the network administrator to make one change that affects many users rather than many changes that affect a single user. As a result of the emphasis on users and groups in Active Directory, this particular category has received special attention from Microsoft.

One of the more important changes that Microsoft has made is to consolidate the plethora of commands that you needed in Windows NT 4 to manage users and groups into a single set of interfaces. You no longer need to use the various Net commands such as NetUserAdd( ) to work with users and groups in Windows 2000. The ability to work with objects greatly reduces complexity of writing management application. In fact, you'll find that users and groups use the same interfaces as any other object in Windows 2000, which is one benefit of using Active Directory.

The way security is handled has also changed. Under Windows NT 4, users and groups were handled differently from system processes. Active Directory treats everyone the same, which actually decreases the complexity of working with security, even though the level of security has increased for this product.

It's not too surprising to learn that both human users and system processes can use the User object to log into the system. It's also important to understand the Contact object from a usage perspective. Though the User object is a security principle, the Contact object isn't. Users can log into the system, but Contacts are limited to reference only. In short, you'd include Contacts within a domain so that your application could reference the contact in a letter or a user could send them e-mail. In other words, when writing an application, you don't always have to create a User object for a human who will be referenced by the system — only those actually logging on need a User object. The Computer object is also part of the "user" picture. A Computer object represents a machine where no user is logged on. It gives the computer the access required to update local information in the background when a user isn't logged into the system. It's also used during the initial user-authentication process to give the computer access to Active Directory for verification purposes.

Groups also have their corollary when it comes to objects. The Organizational Unit could be looked upon as a group-like object. However, there are some significant differences between the Group and Organizational Unit object. For example, an Organizational Unit appears only in one domain tree, where the same Group could be used in multiple domain trees. In addition, while the Group object is normally used to hold User, Computer, and Contact objects, along with some types of system resources, an Organizational Unit can hold all types of subordinate objects, including Groups. Finally, the Group object is normally used for granting access and creating distribution lists. The Organizational Unit object is used for administrative delegation and group policy setting.

There are limitations on groups based on the mode that Windows 2000 is in at the time you access Active Directory. The system starts out in mixed mode, which supports a combination of Windows 2000 and Windows NT 4 domain controllers. Using mixed mode means that you can't create universal security groups, nest security groups within other Group objects, or convert Group objects to Windows 2000 group types.

Windows 2000 does support the nesting of Group objects. This allows you to create master groups that are managed by the network administrator and subgroups that are managed by local workgroup administrators. Using group nesting within your application allows you to grant the minimum rights required to use the application to everyone, and then create smaller groups that have special application rights based on need.

Even though it appears from the MMC snap-in that we looked at in Chapter 19 that Windows 2000 supports only one type of group, there are actually several types of groups that you need to know about from a programming perspective. Each of these groups is used for a different purpose, and each allows you to group both users and resources in different ways. The following list provides a quick overview of the various group types that Windows 2000 supports.

- **Distribution:** Allows you to address a single group of people using a single object. For example, you could use this group type for e-mail. Distribution groups use resources more efficiently and improve system performance.
- **Security:** A superset of the distribution group type that also allows you to assign rights to a group of resources or users. Security groups can appear within ACLs.

Groups also support a scope. The scope limits where the group can be used and therefore, how it can be accessed. The following list provides a quick overview of the various group scopes.

- **Universal:** Can appear on any domain in the forest. Members can be chosen from anywhere in the forest as well.
- **Global:** Can appear on any domain in the forest. However, members must come from the domain containing the group.
- **Domain Local:** Will only appear within the local domain. Members must come from the domain containing the group.

Now that you've gotten some background on the User and Group objects, let's talk about the interfaces that Active Directory provides to work with them. Table 20-2 shows the five interfaces that you'll commonly work with to view, add, delete, or modify both user and group information. Notice that there are only two interfaces that are specific to users and groups. All standard objects within Active Directory have interfaces that fall into the persistent object category (we discuss other categories of interfaces in the "Working with the ADSI API" section of the chapter). The rest of these interfaces are used for other purposes within Active Directory.

Table 20-2
Active Directory User and Group Interfaces

Interface	*Description*
IADs	Provides access to object properties and methods. You can also use this method to determine the object name, class, GUID, Active Directory path, parent, and schema.
IADsPropertyList	Allows you to read, modify, and update property values within an object.
IADsUser	Allows you to read, modify, and update an end user account. You can also use this interface to obtain user statistics like the last time the user logged on, the number of bad login attempts, personal information like home address, and work information like the user's manager. In short, this interface is designed to work with the unique properties of the User object.

Interface	*Description*
IADsGroup	Allows you to read, modify, and update a group account. There are methods for adding objects to and removing objects from the group, along with a method for testing group membership.
IADsMembers	Supports group membership for individual accounts. This interface includes methods for counting the group members, to obtain a new group enumeration, and to view or set filters applied to the group membership.
IDirectoryObject	Provides nonautomation clients (like Visual C++) access to either User or Group objects. Using this interface provides better performance for nonautomation clients, but also means that you need to know more about the attributes associated with individual objects.

Working with the ADSI API

ADSI is the API that allows you to work with Active Directory and allows you to add the new features that Active Directory provides to your applications. Like many other parts of Windows 2000, ADSI relies heavily on COM components, especially when it comes to working with automation clients like Visual Basic. You'll also find that ADSI supports two nonautomation client interfaces for programming languages such as Visual C++, and a plethora of functions designed to make it easier to access various Active Directory services directly.

It's interesting to note that only Windows 2000 products come with Active Directory as part of the package. Microsoft has provided Active Directory support for both Windows 95 and Windows 98, along with a setup for older versions of Windows NT (although which versions are supported isn't clear as of this writing). The Windows NT support includes both Intel and Alpha processor machines. You can find links for all four levels of support at `http://www.microsoft.com/NTServer/nts/downloads/other/ADSI25/`. This site also includes a link for the Active Directory Services Interfaces SDK. If you want more complete ADSI information, then look at the Active Directory Service Interfaces Web site at `http://www.microsoft.com/Windows/server/Technical/directory/adinterface.asp`.

At this point in the chapter, we've discussed a lot of common interfaces and talked about how you can use them to work with Active Directory. However, ADSI provides a lot more in the way of interfaces than what we've covered so far. Table 20-3 contains a list of additional interfaces that we've neither discussed nor covered fully. You'll want to spend some additional time learning about these interfaces as you work with Active Directory using ADSI.

Table 20-3 doesn't contain a complete list of ADSI interfaces. Some of these interfaces are already covered in other areas of the chapter, so they aren't covered again here. In addition, this table doesn't include any of the NDS-specific interfaces. What this table does cover are the more common interfaces that you'll use on a regular basis. Because Active Directory is a moving target at this time, you'll always want to use the latest Platform SDK as your source of information for new features. You can download a copy of the current Platform SDK at `http://msdn.microsoft.com/developer/sdk/platform.asp`.

Table 20-3
Active Directory Automation Interfaces

Interface	*Description*	*Purpose*
Core		
IADs	Defines the basic features, properties and methods, of any ADSI object.	Allows you to learn about the features common to every object such as its name, parent, class, GUID, path, and schema. Special methods allow you to work with either single value or multiple value properties.
IADsContainer	Enables an ADSI container to create, delete, enumerate, and manage ADSI objects.	This is a simple interface that allows you to learn more about the objects held within an ADSI container. You can count the objects, create filters to exclude objects from a count or enumeration, enumerate the objects, move them to another location, or create copy that you can modify.
IADsNamespaces	Allows you to work with namespace objects.	Contains two methods for working with namespaces. The first gets the current namespace container name, whereas the second modifies the namespace container name to a new value. The new namespace container name then becomes the default namespace container for the user.

Interface	*Description*	*Purpose*
Data Type		
IADsLargeInteger	Allows you to work with large (64-bit) integers.	Use this interface to work with 64-bit integers within Active Directory. Since many compilers can't handle 64-bit integers as of this writing, there are two sets of methods required to work with the number. The first set deals with the lower 32-bits, whereas the second set deals with the upper 32-bits.
Dynamic Object		
IADsComputerOperations	Manages computer operations across the network.	Contains two methods. The first allows you to determine the current status of any computer. The second allows you to shut a computer down from a remote location.
IADsFileServiceOperations	Manages file services across the network.	Use this interface to work with the open and active file services sessions for the domain. There are two methods, one that works with resources and another that deals with sessions.
IADsPrintJobOperations	Provides access to the print jobs currently executing on the domain.	This interface contains a variety of methods for obtaining the current status of print jobs. In addition, there are methods for pausing, reordering, and resuming print jobs.
IADsPrintQueueOperations	Provides access to the print queues within the domain.	This interface will allow you to get the status of the print jobs currently residing within the queue. There are also methods for pausing, resuming, and purging print jobs.

Continued

Table 20-3 *(continued)*

Interface	*Description*	*Purpose*
IADsResource	Works with open file service resources.	Allows you to view open file service resource attributes, including user, user path, resource path, and the resource lock count. A file service resource is considered open when the user accesses it and creates a connection to it.
IADsServiceOperations	Works with system services, active or not.	Use the methods in this interface to stop, start, pause, or continue system services on the domain. In addition, you can determine the current system service status and set the system service's password.
IADsSession	Works with active file service sessions.	Allows you to view user statistics for active file service sessions. A session is considered active when a user creates a connection to the file system and opens one or more files. The user statistics include user name, user path, computer name, computer path, the amount of connection time for the session, and the amount of time that the connection has been idle.
Extension		
IADsExtension	Implements the application extension model that allows programmers to create new application types.	Use the methods in this interface to add application specific behaviors into existing ADSI objects. An application extension uses aggregation to modify the behavior of the existing object.

Interface	*Description*	*Purpose*
Persistent Object		
IADsCollection	Manages an arbitrary collection of object.	Collections are essentially specialized arrays that can hold any type of directory services object. There are methods for adding, removing, retrieving, and enumerating objects within the array.
IADsComputer	Allows access to any computer type.	Use the methods in this interface to obtain common information about workstations, servers, or other computer types on the network. We looked at the information provided by this interface in Figure 19-4.
IADsDomain	Represents a network domain and allows you to manage accounts on that domain.	Use the methods within this interface to work with account security. You can determine password length, the age at which passwords are changed, the maximum number of bad logins allowed, lockout parameters, and other password attributes.
IADsFileService	Represents the file service and allows you to manage user access.	Allows you to modify the file service description and the number of users who are allowed to use the file service at the same time.
IADsFileShare	Allows you to modify the attributes of any file share on the network.	Provides the methods required to determine the current user count for a file share. In addition, you can both view and set the file share description, host computer path, shared directory path, and maximum user count.

Continued

Table 20-3 *(continued)*

Interface	*Description*	*Purpose*
IADsLocality	Represents the geographical location of the domain.	Use the interface methods to view or set the locality description, region name, postal code, or "see also" (note) information.
IADsO	Represents an Organization object within the domain.	Allows you to view or set the organization description, locality, postal address, telephone number, fax number, and see also information.
IADsOU	Represents an Organizational Unit object within the domain.	Allows you to view or set the organization description, locality, postal address, telephone number, fax number, see also information, and business category.
IADsPrintJob	Defines individual print jobs within the print queue.	Provides management methods for active print jobs. There are methods that allow you to view the host print queue location, user name, user path, time submitted, total pages, and size. In addition, you can view or set the description, priority, start time, until time, notification value, and notification path.
IADsPrintQueue	Represents a print job destination on the network.	Use the methods in this interface to manage various printer options. You can view or change the printer path, model, data type, print processor, description, location, start time, until time, default job priority, queue priority for processing data on the printer, banner page, print devices, and network addresses.

Interface	*Description*	*Purpose*
Property Cache		
IADsPropertyEntry	Provides the methods necessary to manage attribute values as defined within the schema.	This is an individual entry within a property list. The methods in this interface allow you to view or modify the name, data type, control code, or value of individual attribute values.
IADsPropertyList	Contains one or more property entries that are associated with an object attribute.	Allows you to manage the value entries associated with an object attribute. You can list the entries, obtain a total number of all entries, get or put entries, reset the values within an individual entry or purge the entire list.
Schema		
IADsClass	Manages object descriptions within the schema.	Provides the interfaces for describing the creation of Active Directory objects. This includes the class identifier (CLSID), object identifier (OID), mandatory and optional properties, and help file data.
IADsProperty	Manages object attributes within the schema.	Determines how a property is managed and created by Active Directory. The methods in this interface include the OID, path to the syntax object, minimum and maximum ranges, whether the property is multivalued, and any additional qualifiers needed to fully describe the property.
IADsSyntax	Manages the automation data types used to describe object property values within the schema.	Provides a single method that returns the automation value for a property. These are the virtual types like VT_BOOL (for a BOOL value) used in other parts of Visual C++ to describe the type of a variable or constant.

Continued

Table 20-3 *(continued)*

Interface	*Description*	*Purpose*
Utility		
IADsDeleteOps	Deletes an object from the underlying directory structure.	Contains a single method that an object can call to delete itself from the directory structure once it's no longer needed.
IADsObjectOptions	Allows you to manage provider-specific options for working with ADSI.	Contains two methods for viewing and modifying ADSI provider options. There are four standard options to choose from: server name, referrals, page size, and security mask. Individual providers may support other options.
IADsPathName	Parses Windows and X.500 paths within ADSI.	The methods in this interface allow you to work with paths provided by application programs, users, other directory services, and other objects.

Summary

In this chapter, we talked about the programming considerations for working with Active Directory. You learned that, in a lot of ways, working with Active Directory involves many of the same practices that working with any database requires. Of course, because Active Directory is also used at a system level by the operating system, there are special APIs for getting the job done. Here are some additional conclusions that can be drawn from the contents of this chapter:

- Active Directory is essentially a large database managed by a series of COM components that provide output through an MMC interface.
- Third-party support for Active Directory started long before the product actually appeared on the market. Make sure you stay on top of the new features that Active Directory provides by visiting these third-party sites in addition to the sites that Microsoft provides.

- One of the more important security concepts in Active Directory is per task verification. Even though a user doesn't have to provide his or her credentials each time a task is started, Active Directory does check the active access token against the object's security descriptor before performing the next task.
- MMC normally hides many details of Active Directory from the user. You need to use the View➪Advanced command to reveal these details. Even so, some details are hidden from view and require you to "drill down" through several layers of dialog boxes or menus to find them.
- The three interfaces used to manage security under Active Directory are IADsAccessControlEntry, IADsAccessControlList, and IADsSecurityDescriptor. You may need to use some special interfaces when working with third-party ADSI providers.
- Binding is the process of requesting access to an object, allowing Active Directory to verify your authorization to the object, and then receiving an object reference.
- The two most common functions for binding to an object when using Visual C++ are ADsGetObject( ) and ADsOpenObject( ).
- Objects can be found using either the global catalog or LDAP.
- Using rootDSE allows you to implement serverless binding because it knows the current location of the domain container, schema container, and configuration container.
- You can implement a two-step search-and-access method for Active Directory objects using the IDirectorySearch and IDirectoryObject interfaces.
- Use the NetEnumerateTrustedDomainsEx( ) function to obtain a reference to a domain tree. This will allow you to perform object searches at the domain tree level rather than the forest level.
- Group objects support the concepts of type and scope, both of which determine the type of Group object created and its capabilities.
- There are two essential interfaces for managing groups and users, IADsGroup and IADsUser, along with several helper interfaces.
- The ADSI API provides a wealth of interfaces that you can use to manage your Active Directory setup, including some specialized interfaces that make it easier to work with the NDS directory services.
- Active Directory isn't just a Windows 2000 technology. You can also download versions of Active Directory support for Windows 95/98 and older versions of Windows NT for both the Intel and Alpha platforms.

✦ ✦ ✦